アルゴリズム入門
第5回：ライフゲーム

担当教員：森畑 明昌

morihata@graco.c.u-tokyo.ac.jp

if文のルール（復習）

```
if 条件式:
    なんとか1
else:
    なんとか2
```

- **条件式**を計算してみて、
  成り立つ（true）なら**なんとか1**を、成り立たない（false）なら**なんとか2**を、実行する
  - else なんとか2は省略してもよい

if文の練習（復習）

1つの値をとり、その値が負なら−1を、正なら1を、0なら0を返す関数signを定義せよ

```
def sign(x): #xの符号を返す関数
    if x < 0:
        return (-1)
    else:      #x≧0の場合
        if x > 0:
            return 1
        else:
            return 0
```

ここまでで学んだこと

- 変数の使い方
- 関数の使い方
- `for`文の使い方
- `if`文の使い方
- 配列の使い方
- ライブラリの使い方

これらで本質的にはあらゆるプログラムが書ける。
後は書き方や楽さの問題

前回の宿題

宿題:

配列中から2番目に大きな要素を見つける関数**sndmax**を定義せよ。

ポイント:
1番目に大きい値と2番目に大きい値を両方持っておく

宿題の回答例

```
def sndmax(a):
    if a[0] < a[1]:
        m1 = a[1]      #最大値
        m2 = a[0]      #2番目
    else:
        m1 = a[0]
        m2 = a[1]
    for i in range(2, len(a)):
        if a[i] > m1: #a[i]が最大
            m1 = a[i]
            m2 = m1
        else:
            if a > m2 #a[i]が2番目
                m2 = a
    return m2
```

今週の話題:ライフゲーム

ライフゲーム

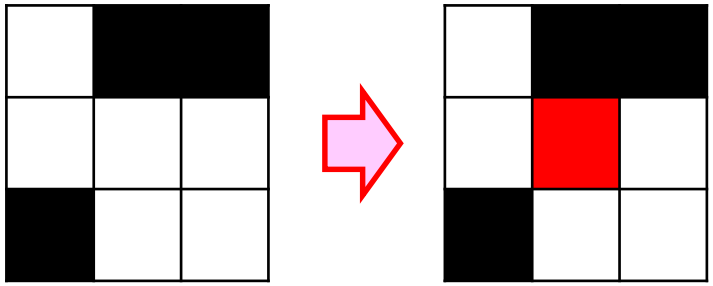
- 数学者が1970年に提案
- ゲーム ＝ ルールに従って局面が進行する
- 大きな動機:生命とは何か？
  - なぜ生命は繁殖するのか？
  - 何故生命は絶滅するのか？
  - それの本質とは何か？　どれくらいシンプルに物事を説明できるか？

ライフゲームのルール

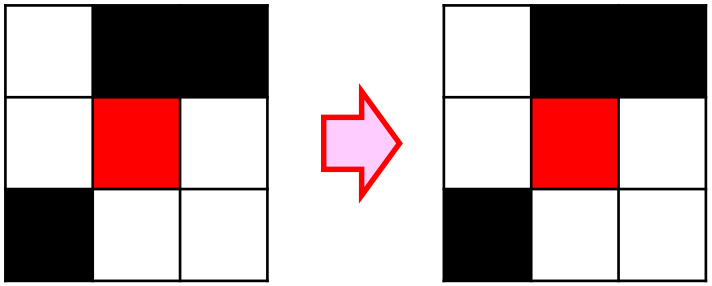
- 格子の上に1種類の「生命」がいる
  - ■：生命、□：空白
- ルールは以下の4つ。全て隣接8セルに依存
  - 生きているセルの場合：
    - 過疎：隣接セルに生命が1体以下なら死滅
    - 過密：隣接セルに生命が4体以上なら死滅
    - 隣接セルに生命が2-3体なら生存継続
  - 死んだセルの場合：
    - 誕生：隣接セルに生命が3体なら生命誕生

絵で見るライフゲーム

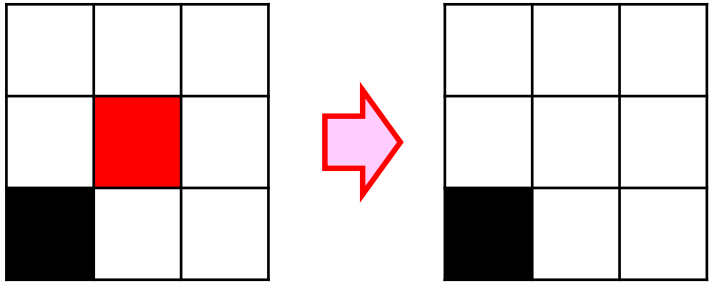
- 誕生



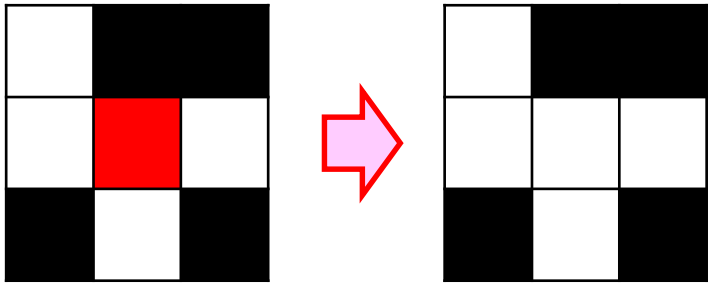
- 生存



- 過疎



- 過密



ライフゲームの影響

ライフゲームは、シンプルなルールだが、初期配置次第で色々なことが起こる

- ある地域に固着したまま動かない生命群
- 周囲へどんどん繁殖してゆく生命群
- 一定の方向・速度で移動してゆく生命群
  - そんな生命群を産み出し続ける生命群

問題:ライフゲームのプログラムを書け

やや複雑なプログラムの書き方

- より単純なケースを考える:
  - あるセルが次の世代でどうなるか？
- 単純なケースの組合せでできないか考える:
  - 全てのセルについて、次の世代でどうなるかをそれぞれ計算すれば良い

1セルでの挙動

```
def lg_rule(cur, neighbor):
    # 1セルの挙動。0が空白、1が生命あり
    # cur:現在の状況、neighbor: 隣接生命数
    if cur == 0: # 現在空白
        if neighbor == 3:
            return 1 # 誕生
        else:
            return 0
    else: # 現在生命有り
        if neighbor == 2 or neighbor == 3:
            return 1 # 生存
        else:
            return 0 # 過疎か過密で死滅
```

補足：複雑な条件

条件を組み合わせて複雑な条件を書ける

- 条件1 **and** 条件2
  - 条件1と条件2が両方とも**True**なら**True**、
    そうでなければ**False**
- 条件1 **or** 条件2
  - 条件1と条件2のどちらかが**True**なら**True**
    そうでなければ**False**
- **not** 条件
  - 条件が**True**なら**False**、**False**なら**True**

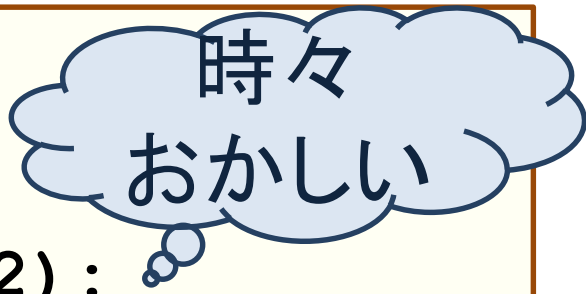
（※）今回は**2 <= neighbor <= 3**でもよい

隣接セルの生命数

次に隣接セルの生命数を数える

- 格子構造は2次元配列（配列の配列）で表す
  ➔ i行j列目はdata[i][j]
- i行j列目の隣接セルの生命数＝
  data[i-1][j-1]～data[i+1][j+1]の和－自分

```
def count_neighbor(data,i,j):
    count = 0
    for k in range(i-1,i+2):
        for l in range(j-1, j+2):
            count = count + data[k][l]
            #周囲の和を求める
    return count - data[i][j]   #自分を引く
```



うまく動かない時の対処法

- どういうときにうまく動かないか、
  どこでうまく動いていないかを調べる
  - 入力を色々試す
  - **print**を挿入して動作をトレース

- 今回：**i**や**j**が小さい・大きい場合におかしい
  - 例：**i=0,j=2**のとき
    ➔ **data[-1][1]**は何かおかしい

（※）実際には**data[-1]**は**data**の最後の要素

正しいプログラム

```
def count_neighbor(data,i,j):
    count = 0
    for k in range(i-1,i+2):
        for l in range(j-1, j+2):
            if 0 <= k < len(data) and
                #kが適切な範囲にある
                0 <= l < len(data[k]):
                #lも適切な範囲にある
                count = count + data[k][l]
    return count - data[i][j] #自分を引く
```

ライフゲーム1世代分

組み合わせれば1世代分できる
配列の要素を変数のように変更できることに注意

```
import arrayUtil

def lifegame_step(data): # ライフゲーム一世代分
    n = len(data)
    m = len(data[0])
    next = arrayUtil.make2d(n,m)   #次世代のデータ
    for i in range(0,n):
        for j in range(0,m):
            c = count_neighbor(data,i,j)
            next[i][j] = lg_rule(data[i][j],c)
            #各点でルールに従い次世代を計算
    return next
```

arrayUtilライブラリ

arrayUtil.py

- 提供している関数:
  - **arrayUtil.make1d(*n*)**
    長さ*n*の配列を作る
  - **arrayUtil.make2d(*m*,*n*)**
    m×*n*の2次元配列を作る
  - **arrayUtil.make2d(*l*,*m*,*n*)**
    l×*m*×*n*の3次元配列を作る

プログラムを作ったら……

プログラムを作ったら:

- 正しさの確認 ＝ テスト
  - どうせなら目で見て挙動を確かめたい（可視化）
- 実行速度の見積もり
  - 格子点それぞれについて、周囲8マスを見て次世代を決める
    ➔ 格子点の数に比例する時間がかかる

plotライブラリ

plot.py

- 今回使うのは:
  - **plot.image_show(*image*)**
    二次元配列***image***を表示。画像の形式は、左下が(0,0)、各点は数値なら白黒（0が白、1が黒）、カラーの場合赤、緑、青の値から成る配列（[1,0,0]が赤、[1,0,1]なら紫など）
  - **plot.animation_show(*image*)**
    「画像＝二次元配列」の配列を受け取り、アニメーションとして表示

とりあえずやってみよう

```
>>> import plot
>>> data = [[0,0,0,0,0,0],
            [0,0,0,0,0,0],
            [0,0,1,1,1,0],
            [0,0,1,0,0,0],
            [0,0,0,1,0,0],
            [0,0,0,0,0,0]]
>>> plot.image_show(data)
>>> plot.image_show(lifegame_step(data))
```

どんな画像が見えた？

世代変化をアニメで見よう

```
def lifegame(data,steps):
    results = arrayUtil.make1d(steps)
    # アニメ用の画像の列を用意
    for i in range(0,steps):
        results[i] = data # 各ステップの結果を格納
        data = lifegame_step(data)
    return results
```

```
>>> images = lifegame(data,20)
>>> plot.animation_show(images)
```

動きが見えた？

練習問題

練習1.

与えられた画像（2次元配列）に、指定された2点を対角とする長方形を指定された色で描くプログラムを書け

練習2.

与えられた画像（2次元配列）の指定された座標に指定された色で指定された半径の円を描くプログラムを書け

練習問題1の回答例＆実行例

```
def draw_rect(image, sy, sx, ey, ex, color):
    # (sx,sy)と(ex,ey)を頂点とする長方形。
    # sx <= ex かつ sy <= ey の必要あり
    for i in range(sy, ey+1):
        for j in range(sx, ex+1):
            image[i][j] = color
    return image
```

カラー画像は
3次元配列

```
>>> image = arrayUtil.make3d(150,200,3)
>>> image = draw_rect(10,20,60,90,[1,0,0]))
>>> plot.image_show(image)
```

練習問題2の回答例＆実行例

```
def draw_circle(image, y, x, r, color):
    # (x,y)を中心とする半径rの円
    for i in range(sy, ey+1):
        for j in range(sx, ex+1):
            if (y-i)**2 + (x-i)**2 < r**2:
                # 中心からの距離がr以下なら
                image[i][j] = color
    return image
```

```
>>> image = arrayUtil.make3d(150,200,3)
>>> image = draw_circle(90,90,50,[0,1,1]))
>>> plot.image_show(image)
```

レポート課題（2回目）

何かの画像を出力するプログラムを作れ

- 16×16以上の大きさの画像にせよ
- 変数や関数、`for`文や`if`文をうまく使おう
- プログラムと画像を見て加点・減点する
  - プログラミングの課題であって、美術の課題ではない。プログラムの内容を主に見る。
  - （プログラミング以外の）独創性にも加点はする。
  - 真っ白や講義そのままの内容、あまりにシンプルなもの等は減点

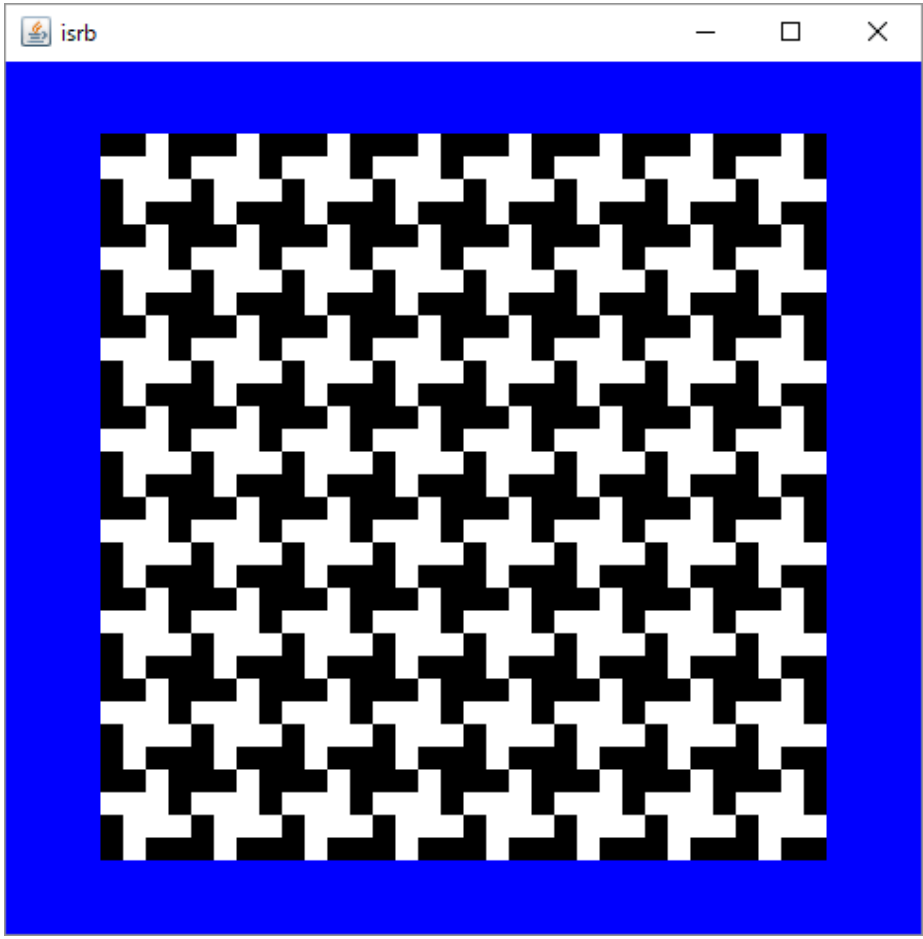
レポート課題（締切と提出方法）

- 締切：11/17（金）22:00
- 提出先：ITC-LMS（締切後はメールで）
- 提出形式：プログラムとその簡単な説明を提出（画像は不要）
- 注意点
  - コピペは不正行為である（見せた方も）
    - 他人のプログラムを「見ながら」書いてもコピペ。
    - 友達と話し合うのは◎、プログラムは自力で
    - 参考資料がある場合は明記すること

プログラム例(1)

与えられた整数nに対し、右のようなn×nの模様を作る関数を作る

ヒント:
規則的な模様なので、座標から何かを計算して白黒を決めれば良さそう



プログラム例(2)

下のような虹模様を描く関数を作る

ヒント:

基本は半円だが、中心からの距離によって色を変える